Amendments to the Specification:

Please replace the title with the following rewritten title:

METHOD AND APPARATUS FOR DYNAMICALLY PROGRAMMING A FIELD PROGRAMMABLE GATE ARRAY (FPGA) IN A COPROCESSOR

Please amend the paragraph beginning at page 1, line 6 as follows:

The use of Field Programmable Gate Arrays (FPGA) on an Application Specific Integrated Circuit (ASIC) chip and/or a system-on-a-chip to provide instruction level hardware acceleration and resource sharing is known in the art. Figure 1 illustrates a conventional apparatus utilizing an FPGA. The apparatus can be an ASIC, a system-on-a-chip, or some other chip comprising a main processor 102, a coprocessor 104, and a local bus, such as the Processor Local Bus (PLB) 106 developed by International Business Machines Corporation™. Other local buses can also be used, such as the Advanced Micro-Controller Bus Architecture (AMBA) developed by ARM™. The coprocessor 104 comprises FPGA cells 110 and a plurality of interfaces. The interfaces include a programming interface 108, through which the FPGA 110 is programmed, a PLB interface 112, and an Auxiliary Processing Unit (APU) interface 114. The APU interface 114 enables special hardware accelerated functions to be tightly coupled to the processor 102 at the instruction flow level. For loosely coupled operations, the processor 102 communicates with the coprocessor 104 via the PLB 106. Fetched instructions inside the processor 102 are simultaneously shared with the coprocessor 104 through the APU interface. The coprocessor 104 signals the processor 102 when it sees a valid instruction, or operations code ("opcode"), for its execution unit. The coprocessor 104 then performs the requested function on operands supplied with the instruction and passes the result back to the processor 102 through the APU interface 114.

Please amend the paragraph beginning at page 3, line 18 as follows:

Figure 1 illustrates a conventional apparatus utilizing an FPGA.

Please amend the paragraph beginning at page 3, line 19 as follows:

Figure 2 illustrates a preferred embodiment of an apparatus for dynamically programming an FPGA in accordance with the present invention.

Please amend the paragraph beginning at page 3, line 21 as follows:

Figure 3 is a flowchart illustrating a preferred embodiment of a method for dynamically programming an FPGA in accordance with the present invention.

Please amend the paragraph beginning at page 4, line 1 as follows:

Figure 4 illustrates a preferred embodiment of a coprocessor with a dynamically programmable FPGA in accordance with the present invention.

Please amend the paragraph beginning at page 4, line 3 as follows:

Figure 5 is a flowchart illustrating in more detail the preferred embodiment of the method for dynamically programming an FPGA in accordance with the present invention.

Please amend the paragraph beginning at page 4, line 7 as follows:

Figure 7 illustrates a second preferred embodiment of the apparatus for dynamically programming an FPGA in accordance with the present invention.

Please amend the paragraph beginning at page 4, line 21 as follows:

Figure 2 illustrates a preferred embodiment of an apparatus for dynamically programming an FPGA in accordance with the present invention. The apparatus comprises a main processor 202, a coprocessor 204, and a Processor Local Bus (PLB) 206 as the local bus. Other types of local buses may be used. As shown in the embodiment of Figure 2, the coprocessor 204 is separate from the main processor 202. The processor 202 comprises an exception subroutine 222, which is described further below. The coprocessor 204 comprises a dynamically programmable FPGA 210, a programming interface 208 for programming the FPGA 210, a PLB interface 212, and an APU interface 214. The apparatus further comprises a memory 216 for storing configuration bit streams 220 for various functions that can be requested by applications executed by the processor 202. The memory 216 can be embedded on a chip along with the processor 202 and coprocessor 204 or external to the chip. According to the present invention, during the execution of an application by the processor 202, if a function requested by the application had not been programmed into the FPGA 210, the configuration bit stream for the function can be fetched from the memory 216, and sent to the programming interface 208 via a programming channel 218 coupled to the PLB 206. The FPGA 210 ~~are~~ is then programmed with the fetched configuration bit stream. Once programmed, the function requested by the application can be performed by the coprocessor 204. In this manner, the FPGA 210 ~~are~~ is programmable "on the fly", i.e., dynamically during the execution of an application.

Please amend the paragraph beginning at page 5, line 17 as follows:

Figure 3 is a flowchart illustrating a preferred embodiment of a method for dynamically programming an FPGA in accordance with the present invention. First, the execution of an application by the processor 202 begins, via step 302. Next, the coprocessor 204 receives an

instruction from the processor 202 to perform a function for the application, via step 304. In the preferred embodiment, the instruction is sent to the APU interface 214 of the coprocessor 204. Next, it is determined that the FPGA 210 of the coprocessor 204 is not programmed with the logic for the requested function, via step 306. In the preferred embodiment, the APU interface 214 returns an error when the FPGA 210 ~~are~~ is not programmed with the logic to perform the function. Then, the processor 202 fetches the configuration bit stream 220 for the function from the memory 216, via step 308. The FPGA 210 ~~are~~ is then programmed in accordance with the configuration bit stream 220, via step 310.

Please amend the paragraph beginning at page 6, line 6 as follows:

In the preferred embodiment, when the processor 202 receives the error from the APU interface 214, the processor 202 initiates the exception subroutine 222, which fetches the configuration bit stream 220 for the function from the memory 216. The bit stream 220 is then sent to the programming interface 208 via the programming channel 218. In the preferred embodiment, the programming channel 218 is a Direct Memory Access (DMA) channel. The exception subroutine 222 of the processor 202 would point the DMA channel to the location in the memory 216 where the configuration bit stream 220 for the requested function is stored. The DMA then fetches the configuration bit stream 220 and writes ~~it~~ the configuration bit stream 220 into the FPGA 210. Other types of programming channels can be used without departing from the spirit and scope of the present invention. For example, Input/Output (I/O) register logic on the PLB 206 can be used, through which the processor 202 can program the FPGA 210 directly.

Please amend the paragraph beginning at page 6, line 17 as follows:

Figure 4 illustrates a preferred embodiment of a coprocessor 204 with a dynamically

programmable FPGA in accordance with the present invention. The coprocessor 204 comprises a plurality of coprocessor state machines (CSM1-CSM3). Each coprocessor state machine (CSM) represents a function type and manages one or more coprocessor managed resources (CMR1-CMR9). The coprocessor managed resources (CMR) represent the logic inside the FPGA 210 for each coprocessor function managed by its corresponding CSM. An instruction received by the APU interface 214 describes the instruction type and may contain additional information or parameters. The APU interface 214 forwards the instruction to the appropriate CSM based on the instruction type. The CSM then executes the appropriate CMR for the requested function. If none of the CMRs were programmed with the requested function, the APU interface 214 returns an error to the processor 202. The configuration bit stream for the requested function is fetched, and the appropriate CMR is programmed with the configuration bit stream, as described above.

Please amend the paragraph beginning at page 7, line 8 as follows:

For example, assume that the application issues an instruction to the coprocessor 204, requesting that a specific event timer function be performed, via step 302. The APU interface 214 receives the instruction and ~~forward it~~ forwards the instruction to ~~CMR1~~ CSM1, which manages event timer functions. CSM1 ~~CMR1~~ determines that the logic for the requested event timer function has not been programmed into any of CMR1-CMR4, via step 306. The APU interface 214 returns this error to the processor 202. The processor 202 then executes the exception subroutine 222. The exception subroutine 222 fetches the configuration bit stream for the requested event timer function from memory 216, and sends the configuration bit stream to the programming interface 208. One of the CMR's is then programmed with the appropriate configuration bit stream.

Please amend the paragraph beginning at page 7, line 18 as follows:

Figure 5 is a flowchart illustrating in more detail the preferred embodiment of the method for dynamically programming an FPGA in accordance with the present invention. During execution of an application, the processor 202 fetches an instruction, via step 502. The instruction is sent to the APU interface 214, via step 504. If the instruction is for the coprocessor 204, the APU interface 214 issues a commit, via step 506. If not, the APU interface 214 processes the next instruction on the processor pipeline, via step 508. If the APU interface 214 commits, then the APU interface 214 attempts to execute the function. If the function has been programmed into the FPGA 210, then the FPGA execute the function, via step 516. If the coprocessor 204 can then respond immediately with a value in the results register, via step 518, then the result is returned, via step 522, and the instruction completes, via step 524. If the results cannot be returned immediately, then the function requires more clock cycles to complete, and the processor pipeline is held for the result, via step 520.

Please amend the paragraph beginning at page 8, line 13 as follows:

Figure 6 is a flowchart illustrating in more detail the programming of the FPGA with the desired function in accordance with the present invention. If the function has not been programmed into the FPGA 210, the coprocessor 204, and more specifically the APU interface 214, provides an exception code in a results register, via step 602. The processor 202 responds to the exception code by branching to the exception subroutine 222, via step 604. The exception subroutine 222 decodes the function identifier passed to it, via step 606. The exception subroutine 222 then requests and is granted ownership of the required function by the programming channel 218 across the PLB 206, via step 608. The processor 204 then fetches the

configuration bit stream 220 for the function from the memory 216, via step 610. The exception subroutine 222 identifies the exception type and the coprocessor instruction type, via step 612. The exception subroutine 222 ~~It~~ then performs a sequence of load/store instructions via the programming channel 218 and the programming interface 208 to program the FPGA 210 with the configuration bit stream, via step 614. The processor 202 then reissues the instruction, via step 616.

Please amend the paragraph beginning at page 9, line 9 as follows:

Figure 7 illustrates a second preferred embodiment of the apparatus for dynamically programming an FPGA in accordance with the present invention. In this embodiment, a plurality of processors 702, 708, and 712 share system resources, such as program memory (not shown) and coprocessors 704, 710, and 714. Each coprocessor 704, 710, and 714 acts as a Shared Resource Agent and each is attached to the PLB 206 through a Shared Resource Interface (SRI) 706, 710, and 716, respectively. Coupled to the PLB 206 is a Shared Resource Manager (SRM) 718. The SRM 718 guarantees data consistency is maintained between the shared resources by making sure the right to modify a given resource is only given to one processor at a time. Each processor 702, 708, 712 and coprocessors 704, 710, 714 can independently access the shared system resources through their respective SRI 706, 710, 716, with their access controlled by the SRM 718. The programming of the FPGAs in each of the coprocessors 704, 710, 714 is performed in the same manner as described above, except the SRM 718 can serve as the programming channel 218, and the instructions fetched by a processor 702, 708, 712 can be sent to the FPGAs (not shown) of any of the coprocessors 704, 710, 714. The SRM 718 can program a fetched configuration bit stream 220 into ~~the~~ an FPGA ~~210~~ associated with coprocessors 704, 710, 714. Alternatively, ~~the~~ each processor 702, 708, 712 can request a resource from the SRM

718. After the SRM 718 grants the resource to the a given processor, the processor 202 programs the an appropriate FPGA.

Please amend the paragraph beginning at page 10, line 4 as follows:

Another exception condition can occur when the requested function must be loaded into a coprocessor but there is not enough unused logic resources in the FPGAs associated with the coprocessor to program the new function. In this situation, a Least Recently Used (LRU) algorithm can be implemented either in the coprocessor or as a thread running on the processor. A LRU function manager keeps track of the functions loaded into the coprocessors and how often each has been used in a given period of time. This LRU function manager can be queried to determine which old function can be disabled to free up logic resources to load the new function. In this way, the coprocessor can act as a cache of functions for the processor. This library of functions is managed by the SRM 718. The exception processing of the processor is the same as described above except for the added delay caused by the LRU function manager operation.

Please amend the paragraph beginning at page 10, line 15 as follows:

A method and apparatus for dynamically programming an FPGA have been disclosed. The method includes beginning an execution of an application by the processor; receiving an instruction from the processor by the coprocessor to perform a function for the application; determining that the FPGA in the coprocessor is not programmed with logic for the function; fetching a configuration bit stream for the function; and programming the FPGA with the configuration bit stream. In this manner, the FPGA is programmable "on the fly", i.e., dynamically during the execution of an application. Logic flexibility and space savings on the

chip comprising the coprocessor and processor are provided as well.